Deep Reinforcement Learning with Hand-crafted Features to Solve P-median Problems

En-Cheng Chang[1] , **Paula Carroll**[1] , **Deepak Ajwani**[1]
[1]University College Dublin
en-cheng.chang@ucdconnect.ie, {paula.carroll, deepak.ajwani}@ucd.ie

Abstract

A p-median is a combinatorial optimisation problem with applications to rich real-world domains, such as transportation and telecommunications. As such, it has attracted researchers' attention in optimisation, machine learning and network design domains for decades. This problem has traditionally been solved using exact/approximation algorithms and/or heuristics. However, since this combinatorial optimisation problem is *NP-hard,* exact algorithms suffer from high computational costs while designing heuristics still requires specialised knowledge and considerable design time. In recent years, machine learning techniques have been found to be effective in a range of combinatorial optimisation problems. In this work, we investigate if a deep reinforcement learning approach can achieve efficient and optimal solutions for this problem. The key challenge in utilising reinforcement learning for solving combinatorial optimisation problems is to find an effective state representation. Traditionally, graph embeddings have been used for this purpose, but these techniques have struggled to achieve the optimality-efficiency trade-off required for real applications. In this work, we explore if a state representation based on (i) an embedding based on solution sets and topological views, (ii) carefully engineered hand-crafted features or, (iii) concatenating the embedding and hand-crafted features as states, can help achieve a better optimality-efficiency trade-off. Our experimental results suggest that this is a promising research direction.

1 Introduction

A p-median problem is a subset of *NP-hard* combinatorial optimisation problems (COPs) that models and solves real-world applications in domains such as hub-location selection, transportation and telecommunications. It is also used as a popular clustering technique in machine learning. Formally, a p-median problem can be described in terms of the integer programming formulation (1). Let G be a bipartite graph with two partitions (F, C), where F is the set of facilities and C is that of customers. Also, let p be a positive number that bounds the number of opened facilities. Let y_i denote a facility, x_{ij} represent an edge between a customer node j and y_i and, c_{ij} be the cost of delivering products through the edge x_{ij}. The aim of formulation (1) is to minimise the cost of serving all the customer nodes in the set C under the constraint that at most p facilities are opened.

$$
\begin{aligned}
\min \; & \sum_{i \in F, j \in C} c_{ij} x_{ij} \\
\text{s.t.} \; & \sum_{i \in F} x_{ij} \geq 1, j \in C \\
& x_{ij} \leq y_i, i \in F, j \in C \\
& \sum_{i \in F} -y_i \geq -p \\
& x_{ij} \in \{0,1\}, i \in F, j \in C \\
& y_i \in \{0,1\}, i \in F
\end{aligned}
\tag{1}
$$

The problem has traditionally been solved using approximation algorithms (e.g. [6; 1; 5]) and/or heuristics (e.g. [2]).

In recent years, machine learning (ML) techniques have been found to be effective in a range of combinatorial optimisation problems (e.g. [4; 8; 3; 7; 9]. Reinforcement learning (RL) techniques have been particularly popular for solving COPs such as minimum vertex cover and graph colouring problems (e.g. [3; 7; 9]) as they do not require labelled training data beforehand, but instead rely on a reward function which can be designed using the objective function of the COPs. In this work, we explore deep RL techniques for the p-median problem. Specifically, we investigate if a deep RL approach can be designed to enable the computation of near-optimal solutions for this problem efficiently. The key challenge in utilising RL for solving COPs is to find an effective state representation. In previous literature [7], graph embeddings have been used as a state representation, but these embeddings are often agnostic to the underlying COPs and hence, RL approaches based on embeddings often struggle to achieve the optimality-efficiency trade-off required for real applications. In this work, we explore if a state representation based on (i) an embedding based on solution sets and topological views, (ii) carefully engineered hand-crafted features or, (iii) concatenating the embedding and hand-crafted features as states, can help achieve a better optimality-efficiency trade-off. If successful, such an approach will allow a deeper integration of features from algorithm design literature into ML techniques to efficiently solve a range of COPs.

Our experimental results suggest that this is indeed a promising research direction. Firstly, we find evidence that, in solving harder p-median instances, the usage of carefully engineered features from algorithm design literature greatly assists deep RL techniques in achieving a better optimality-efficiency trade-off compared to the graph embedding based deep RL approaches. Secondly, our three approaches show that they can learn policies from diverse points of views, which turn out to be integratable and lead to better solutions for p-median problems.

2 Methodology

We first modify the deep RL architecture of Dai et al. [7] to suit the p-median problem. To achieve a more efficient training, we fix the subset of parameters in its graph embedding. Besides, we give an extra term for RL states to enable our RL architecture to learn the number of facilities that can be selected.

Then, we propose another deep RL architecture based on a carefully engineered feature set. Thirdly, considering if these two architectures can solve p-median problems, we can rich our RL states by concatenating both expressions of the states and to observe whether the performance improves.

All of our architectures are based on Dueling Deep Q Networks (DDQNs) but using three different state representations. The first approach uses an untrained fixed parameter set to blend information within nodes while the second approach uses hand-crafted features for the node representation. The hybrid representation concatenates the two representations to observe whether a richer representation benefits solution quality. The representations of graphs are fed into deep learning models and, the models return Q-values of the action set. RL then continues to take an action and update the corresponding Q-values.

2.1 Reinforcement learning formulation

RL techniques have demonstrated their advantages in solving COPs as heuristics (e.g.[3; 7; 9]). They omit the need of labelled data but require clear definitions to state representations, actions, rewards and termination criteria. Here, we provide the definitions of these terms in the following subsections. Note that, as described in the $Introduction$, we aim at discovering suitable state representations for solving p-median problems; hence, our approaches are given different state representations but share the same definitions of actions, rewards and termination criteria.

- ***S*tates:** In the first architecture with graph embedding, the neighbours of facility and customer nodes are embedded using a pre-determined fixed parameter set. The neighbouring embeddings are used to update a node's embedding value at every state. In the feature-based model, carefully engineered features are created for every node in a graph. The features are updated whenever a new facility is selected. Similarly, the hybrid model concatenates the two sets of the representations for a node at every state.
- ***A*ctions:** Let S denote a solution set of a p-median problem. The candidate actions to be considered are those not in the set S. Specifically, RL selects a facility from the facility set F in formulation (1) at every state. This facility will be stored into the set S, and the facility should not be selected again in the following states. As a result, the action set is variable-sized.
- ***R*ewards:** We assign 1 to x_{ij} (c.f. ILP formulation (1)) iff the facility y_i in the set S is the closest facility to the customer node j . Computing the objective values of the forumation (1) with this assignment, we can determine the difference in the objective values before and after taking an action. Specifically, we define the rewards as Equation (2) shows, where x_{ij} and x'_{ij} refer to the assignments before and after taking an action. Typically, the reward will be a negative value. With this definition of the reward function, the reward will equal to the negative objective value of the ILP forumation (1) at the final state. Note that in order to bound the number of selected facilities, we also introduce a large penalty to the reward if $\sum_i y_i > p$. We set the penalty to -10^4 as this is significantly smaller than the negative objective values of our experimental instances, and our RL models can learn that an infeasible solution leads to a significantly worse reward.

$$Reward = \sum_{i \in F, j \in C} c_{ij}(x_{ij} - x'_{ij}) \tag{2}$$

- ***T*ermination Criteria:** According to the assignment in $Rewards$, a customer node j is only supplied by a facility in S iff the facility y_i is the closest facility to j. Therefore, if the customer nodes of a p-median problem are supplied but $|S|$ is less than p, opening a new facility will not cause a higher objective value. To sum up, an epoch terminates when all the customers are served and the number of open facilities is greater or equal to p.

2.2 GNN based deep RL

Our first architecture is based on the structure proposed by Dai et al. [7], in which, the embedding process enables nodes to blend information from its neighbouring nodes by a neural network with activation functions. However, the parameters are pre-determined in our architecture and, if the embedding parameters are not learnable, giving activation functions will lead to the majority of the embeddings being transformed into a constant. As such, under our framework, we suggest that activation functions in the graph embeddings should be removed as Equation (3) shows. The Q-values of the actions at a state are estimated using Equation (4). The parameters $\theta_1, \theta_2, \theta_3$ and θ_4 in Equation (3) are neural network parameters. The parameters θ_5, θ_6, θ_7 and θ_8 in Equation (4) are parameters for predicting Q-values. In this approach, we initially build a fundamental GNN model with the graph embedding, where the parameters $\theta_1, \theta_2, \theta_3$ and θ_4 are given

Facility	Deg.	1 Per.	25 Per.	50 Per.	75 Per.	99 Per.	NUC	WNUC	p	MCD
F1	2.00	10.02	10.50	11.00	11.50	11.98	2.00	22.00	2.00	1.00
F2	1.00	7.00	7.00	7.00	7.00	7.00	1.00	7.00	2.00	2.00
F3	1.00	20.00	20.00	20.00	20.00	20.00	1.00	20.00	2.00	1.00

Table 1: Features at 1_{st} iteration for Figure 1. Here, Deg. refers to the degree of the facility node, q Per. denotes the q percentile of incident edges, NUC refers to the number of neighbouring customers of the facility that are not assigned to any other facility before, WNUC refers to the weighted sum of distances between those customers and the facility, p refers to the constraint p of a p-median problem and MCD refers to the minimum degree of the unassigned customer neighbours.

randomly at first and will not be changed anymore, while $\theta_5, \theta_6, \theta_7$ and θ_8 are learned through training. We want to explore if RL can estimate the Q-values of a state that has been encoded by a fixed set of parameters.

$$\mu_v^{(t+1)} \leftarrow (\ \theta_1 z_v + \theta_2 \sum_{u \in N(v)} \mu_u^{(t)} + \theta_3 \sum_{u \in N(v)} \theta_4 c_{uv}) \tag{3}$$

$$value = \theta_5^T relu([\theta_6 \sum_{u \in V} \mu_u^{(T)}, \theta_7 \mu_v^{(T)}, \theta_8(p - |S|)]) \tag{4}$$

Algorithm 1 presents the pseudo-code of the fixed-parameters GNN with RL. Specifically, in Algorithm 1, facility nodes and customer nodes are initialised as follows: v represents a facility/customer node. z_v is a binary variable that denotes if a node has been selected/supplied; therefore, all z_v are initialised as 0. $u_v \in \mathbf{R^d}$ is an embedded vector of a node and is initialised as the same arbitrary values. c_{uv} is the weight of an edge which corresponds to the cost of an edge in formulation (1), and $N(v)$ denotes the incident nodes of a node v. $\theta_1 \in \mathbf{R^d}, \theta_2 \in \mathbf{R^{dxd}}, \theta_3 \in \mathbf{R^{dxd}}$ and $\theta_4 \in \mathbf{R^d}$. To clarify , $\mathbf{R^d}$ refers to a latent of d-dim. In our fixed-parameter GNN, we implement the algorithm with d-dim set to 64, 128, 256. The results of these settings are relatively close; hence, the d-dim is set to 64 in the end to lower computation requirements. $\theta_1, \theta_2, \theta_3$ and θ_4 are pre-given as mentioned, and the embedding will run for T times. Using Equation (3), we will obtain the embedded vectors for each facility node and proceed to estimate their Q-values by Equation (4). Here, $[.,.]$ is the concatenation operator. Equation (4) concatenates (i) the summation of all nodes' embeddings, (ii) the embedding of a node and (iii) the difference between p and $|S|$. The $relu$ operator, $relu() = max(0, v)$, is applied element-wise to all the concatenated elements. As such, $\theta_5 \in \mathbf{R^{3d}}, \theta_6 \in \mathbf{R^{dxd}}, \theta_7 \in \mathbf{R^{dxd}}$ and $\theta_8 \in \mathbf{R^d}$.

Algorithm 1 fixed-parameter DDQN for p-median problems

```
1:  Initialise(Memory:$M$, $\Theta_{action}$, $\Theta_{target}$, counter: $w$)
2:  for $k \leftarrow 1$ to $N$ do
3:      Graph $G \leftarrow D(n, k)$
4:      for epoch $n \leftarrow 1$ to $L$ do
5:          for steps $t \leftarrow 1$ to $Stop$ do
6:              for 1 to $T$ do
7:                  $\mu_v^{(T)} \leftarrow$ Equation (3)
8:              end for
9:              $value\ v_t \leftarrow$ Equation (4)
10:             $a_t \leftarrow$ pickAction($v_t$)
11:             $s_{t+1} \leftarrow$ step($s_t$, $a_t$)
12:             $r_t \leftarrow Reward(s_t, s_{t+1})$
13:             Add ($s_t$, $a_t$, $r_t$, $s_{t+1}$) to $M$
14:             Sample batch $B \overset{iid}{\sim} M$
15:             Update $\Theta_{action}$
16:             if $w\%100 == 0$ then
17:                 $\Theta_{target} \leftarrow \Theta_{action}$
18:             end if
19:             $w$ += 1
20:         end for
21:     end for
22: end for
```

2.3 Feature-based model

Our second architecture uses a set of features to represent a node (with the aggregated node representations forming the state representation). These features for the facility nodes have been carefully selected from the algorithm design and heuristics literature on the p-median problem. These features for the facility node corresponding to the variable y_i include (i) the degree of the node, (ii) 1%, 25%, 50%, 75%, 99% percentile of the incident edge weights, (iii) the number of neighbouring customers of the facility that are not assigned to any other facility yet (iv) the sum of distances between the unassigned neighbouring customers and the facility, (v) the constant p and (vi) the minimum degree of the unassigned neighbouring customers. Note that the last feature captures the intuition that if a customer is only served by one facility, and then that facility needs to be selected.

Figure 1 shows an instance with 3 facility nodes and 3 customer nodes. Assuming $p = 2$, Table 1 presents the feature values of each facility in the first iteration.

Our feature-based RL architecture shares the same pseudo-code structure with Algorithm 1; nevertheless, instead of using graph embeddings to represent a node for value estimation, we use notation h_v to denote the feature set of a node and apply h_v to estimate $value\ v_t$ in Algorithm 1 Line 9.

2.4 Hybrid model

Our third model is a hybrid model blending both fixed-parameter and feature-based models. Considering that the graph embeddings obtained from the fixed-parameter model capture the coherence of neighbouring nodes and can func-

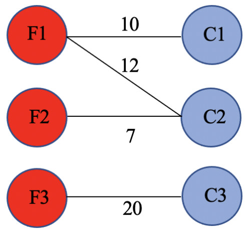

Figure 1: Example of a p-median instance

tion properly as node representations, it is a suitable feature set to be included. Therefore, this hybrid model computes the graph embedding as a set of features for every node at every iteration. It then concatenates the embeddings from section (2.2) and the features from section (2.3) to estimate the Q-value of the facility nodes. We show the Q-value estimation of the hybrid model in Equation (5), where $\theta_9 \in \mathbf{R^{10xd}}$ as there are 10 features for a node and $\theta_9 \in \mathbf{R^{4d}}$.

$$\begin{aligned} value = \theta_1 0^T relu([\theta_6 \sum_{u \in V} \mu_u^{(T)}, \theta_7 \mu_v^{(T)}, \\ \theta_8(p - |S|), \theta_9 h_v]) \end{aligned} \tag{5}$$

3 Experimental Evaluation

To test the efficacy of our approaches, we use a synthetic dataset with one implanted solution (that may or may not be optimal). Our algorithms and experiments are implemented and executed using Python 3.8 and Pytoch 1.7.1 in an environment of Quad-Core Intel Core i5 CPU with 8 GB RAM. The codes are public on Github[1].

Dataset: The synthetic instances for the p-median problem are generated as follows: Given $2n$ and p, the generator produces an instance with n facility nodes and n customer nodes. We select a random subset F_p of p facilities and connect each of these facilities with a random subset (without replacement) of $\lfloor \frac{n}{p} \rfloor$ customers (last facility is connected with remaining customers) with weights $c_{ij} \overset{iid}{\sim} U(a_1, b_1)$. This creates an implanted solution that may or may not be optimal. The remaining facilities (set F_p') have a uniformly random degree between 1 and n with mode 0 and uniformly between $\lceil \frac{n}{p} \rceil$ and $\lceil 2 * \frac{n}{p} \rceil$ with mode 1. These facilities are connected with a random subset of customers with replacement and the edge weights are selected through $c_{ij} \overset{iid}{\sim} U(a_2, b_2)$. If a_1 and b_1 are much smaller than a_2 and b_2 and mode is 0, then it is very likely that F_p is an optimal solution. On the other hand, if $a_1 = a_2$ and $b_1 = b_2$ and/or the model is 1, the set F_p is unlikely to be optimal.

Note that this instance generator is likely to produce non-metric instances which are particularly hard cases of the p-median problem. Many of the heuristics and approximation algorithms for this problem work well on metric instances, but not on non-metric instances.

[1]https://github.com/CharlesEnCheng/IJCAI-DDQN4P-median

3.1 Experiments

Next, we describe our experiments on three different synthetic instance distributions and compare the three proposed architectures to evaluate their efficiency-optimality trade-off. All three experiments are trained with small instances using n randomly picked between 7 and 9 (14-18 nodes for a graph) and p between 2 and $n-1$. Every instance is replayed 100 times, and the model will be evaluated by 100 validation instances and test instances. Specifically, after a model is trained over every 10 graphs for 100 times, we examine the efficacy of the model. Meanwhile, when the examined model reaches a higher feasible rate for the whole validation set, or the model reaches the same feasible rate but obtains a better average approximation ratio to optimal (App/Opt) shown in Equation (6), the model will be saved to examine its performance on the test set at the end of training. Here, $valid$ refers to the validation set; G is a graph in the set and, App_G is the objective value searched by our model and Opt_G is the optimal objective value found by FICO Xpress Optimization tool.

$$App/Opt = \frac{\sum_{G \in valid} \frac{App_G}{Opt_G}}{|valid|} \tag{6}$$

The goal of the three experiments is to explore if any of our reinforcement learning approaches can (i) distinguish between facilities in F_p and facilities in F_p' (ii) which approach achieves a better optimality-efficiency trade-off, (iii) generalise to instances larger than training data and (iv) learn policies from diverse points of views. To observe (iii), the validation and test data will be set larger than the training graph. And for (iv), we create a mechanism ($Best$ thereafter) which picks the solution that is feasible and with the smallest App/Opt out of the three proposed models. If $Best$ yields better numerical results in either the number of feasible instances or App/Opt, it reflects that different models learn diverse policies to solve the p-median problems.

Experiment 1 - Easy: Our first instance distribution is obtained by setting the parameters a_1, b_1, a_2, b_2 and $mode$ as 20, 50, 100, 400 and 0, respectively. For this instance distribution, F_p is likely to be the optimal solution to Equation 1 and, the weights of F_p are significantly smaller than F_p'. The validation and test instances are $2n$ = 30 and p = 5.

The left three sub-figures in Figure 2 show the results of Experiments 1. Here, the x-axis refers to the number of training graphs. As the validation is run every 10 training graphs; hence, 400 in the x-axis represents training on 4,000 graphs and each of the graphs is replayed for 100 times. The left-hand-side y-axis is the number of feasible instances out of the 100 validation instances. Reaching the top shows that a model finds feasible solutions for all the validation instances, and the solid line reflects the number of feasible instances. On the other hand, the right-hand-side y-axis is for App/Opt, and the dash line is the corresponding line. The dash line approaching 1.0 reflects a better solution quality found by an approach. Note that we only compute the average App/Opt for the feasible cases, so the dash line would vanish in some circumstances, such as when a model cannot obtain a feasible

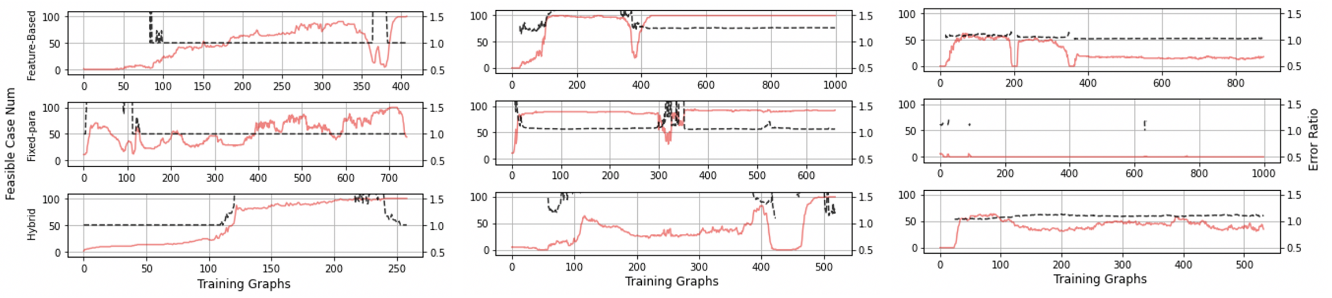

Figure 2: The left three figures are the results of Experiment 1. The top sub-figure is from the feature-based model while the middle one is from fixed-parameters GNN and, the bottom one is from the hybrid model. The middle three are the ones of Experiment 2, and the right ones are the results of Experiment 3.

		Valid					*Test*				
Exp	Model	FN	Opt	App	App/Opt	IFAvgK	FN	Opt	App	App/Opt	IFAvgK
Easy	Feature	100	-514.55	-514.55	1.00	nan	100	-518.03	-518.03	1.00	nan
	GNN	100	-514.55	-514.55	1.00	nan	099	-517.53	-517.53	1.00	6.0
	Hybrid	100	-514.55	-514.55	1.00	nan	100	-518.03	-518.03	1.00	nan
	Best	100	-514.55	-514.55	1.00	nan	100	-518.03	-518.03	1.00	nan
Medium	Feature	100	-203.22	-263.10	1.30	nan	100	-201.20	-266.48	1.33	nan
	GNN	94	-201.44	-213.21	1.06	8.2	97	-200.87	-212.38	1.06	9.3
	Hybrid	100	-203.22	-233.21	1.15	nan	100	-201.20	-231.28	1.15	nan
	Best	100	-203.22	-216.37	1.06	nan	100	-201.20	-213.56	1.06	nan
Hard	Feature	62	-468.39	-498.05	1.06	6.2	46	-462.22	-498.02	1.08	6.2
	GNN	6	-463.50	-510.50	1.10	7.6	10	-472.70	-498.20	1.06	7.6
	Hybrid	64	-465.55	-491.23	1.06	6.1	54	-464.63	-493.72	1.06	6.1
	Best	77	-467.92	-487.66	1.04	6.0	64	-465.80	-488.56	1.05	6.0

Table 2: Comparison between our deep RL approach (denoted Feature) based on carefully engineered features, the deep RL approach based on graph embedding denoted as GNN, the deep RL approach based on both features and embeddings denoted Hybrid and, the model selects solutions from the three models denoted as Best.

solution for any instances or the average App/Opt is greater than 1.5.

The results show that all three algorithms are able to distinguish between the facilities in the two sets and achieve optimal solutions for all the validation instances. The feature-based model reaches optimal after training over 4,000 graphs; fixed-parameter GNN - 7,500 graphs; hybrid model - 2,500 graphs. The hybrid model is with richer features and, requires less training graphs compared with the other two approaches. Further numerical details of the results on validation data and test data are shown in Table 2.

In Table 2, FN represents the number of feasible validation/test instances solved by a RL model; Opt refers to the mean optimal objective values of the feasible instances, and App refers to the mean objective values of that found by our deep reinforcement learning approaches. The computational time of Xpress solving these problems and our approaches are close; nevertheless, our approaches solve p-median problems in polynominal time. Therefore, we hypothesise that these approaches show advantages in speed when the number of nodes increases. IFAvgK shows the average number of test instances on which the reinforcement learning results in an infeasible solution (i.e., a solution with more than p facilities). IFAvgK returns nan when there is no infeasible solution.

Both feature-based and hybrid model get optimal solutions for all validation and test instances while fixed parameter satisfies all validation data but misses one test instance. Note that the one missed test graph is still close to feasible solution according to the near-p IFAvgK.

Experiment 2 - Medium: Next, we mix the edge weight distributions by setting the parameters a_1, b_1, a_2, b_2 and $mode$ as 20, 50, 10, 30 and 0. As the degree of F'_p ranges from 1 to n and the weights are smaller, F_p set is a feasible solution but is unlikely to be optimal. The 100 validation and test instances are n = 15 and p = 5.

Figure 2 shows that our feature-based reinforcement learning model reaches feasibility for 100 validation instances after training over 4,000 graphs and the App/Opt fluctuates at 1.29; GNN reaches near-optimal solutions which is at 1.06 for 94 instances; the hybrid model meets feasibility for all validation instances and with App/Opt around 1.15. Table 2 shows a similar outcome from test data. Note that, from Experiment 2, we find it interesting that different models provide different policies in solving p-median problems. For example, in Table 2, our fixed-parameter GNN is capable of generating better solutions but cannot promise feasibility to all instances.

Besides, the infeasible solutions are severely infeasible. On the other hand, the hybrid model contributes to the feasibility of solutions but the solution quality is less competitive compared with the GNN. $Best$ in this experiment benefits in giving better solution quality where all solutions are feasible and, average App/Opt reaches 1.06. It gives us an insight that using different features to build deep learning RL models leads to various policies. These policies enable models to take advantage in solving specific types of instances.

Experiment 3 - Hard: In this setting, we set the parameters a_1, b_1, a_2, b_2 and $mode$ to be 20, 50, 20, 50 and 1. This experiment sets all the edge weights from the same distribution and the degree distribution is fairly similar. Thus, it is a hard instance for RL to solve. The results are shown in Figure 2 right-hand-side. Both the feature-based model and hybrid model witness an improving trend at first but, the former drops in the end and the latter fluctuates soon. On the other hand, fixed-parameter GNN shows a low FN-trend since the beginning and is unable to learn a functional policy during the whole training. In Table 2, all three models yield similar and small values of App/Opt. Both the feature-based and hybrid models have higher chances to meet the feasibility with around 50 out of the 100 instances. Although there are 46 and 54 percent of instances that are infeasible, the IFAvgK of them are 6.2 and 6.1 respectively, which implies that, these infeasible solutions are fairly close to feasible solutions.

Comparing with Experiment 2 and 3, for test data, $Best$ increases the number of feasible instances from 54 up to 64. 16 out of these 64 are picked from the feature-based model; 6 from the GNN; 34 from the hybrid model and there are at least two models generating the same solutions to the 8 out of 64 instances. Accordingly, although the hybrid model is based on the feature-based and GNN, there are still room and values for integrating these models.

Besides, $Best$ greatly increases the number of feasible instances, which is surprising as the hybrid model is based on the feature-based model and fixed-parameter GNN, and it outperforms these two in meeting feasibility. It proves that the two models learn to solve p-median problems in other ways and may be able to benefit learning solving COPs similar to bagging approaches.

4 Conclusion

From these three experiments, we discover that fixed-parameter is capable of solving simpler p-median problems and can achieve near-optimal solutions. Although it struggles while solving harder problems, the embeddings it provides to the hybrid model assist the model to outperform the feature-based model in solution quality. Besides, even though the feature-based model and fixed-parameter GNN simply hold a subset of information of the hybrid model, they still learn various policies for solving a p-median problem, which can externally enable more instances to be solved. We are thus curious about the potential that various models with different subsets of features would enable RL models to better learn solving COPs as bagging approaches do. Finally, we conclude that (i) purely graph embedding based approaches struggle to reach the same trade-off, neither solve harder instances, while (ii) well-defined features can significantly assist the deep reinforcement learning approach to find approximated solutions using fewer training data. Moreover, (iii) concatenating the hand-crafted features and embedding as states not only requires less training data but also show advantage in meeting feasibility of p-median problems. This gives us evidence that our approach combining algorithm engineering knowledge of good features with a deep reinforcement learning technique has the potential to provide efficient solutions of p-median problems over many different instance distributions. We hypothesise that this approach should work for a large number of other combinatorial optimisation problems as well.

Acknowledgement: This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant number 18/CRT/6183. For the purpose of Open Access, the authors have applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

References

[1] S. Arora, P. Raghavan, and S. Rao. Approximation schemes for euclidean k-medians and related problems. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 106–113, 1998.

[2] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on computing*, 33(3):544–562, 2004.

[3] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[4] Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 2020.

[5] M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *Journal of Computer and System Sciences*, 65(1):129–149, 2002.

[6] M. Charikar and S. Li. A dependent LP-rounding approach for the k-median problem. In *International Conference on Automata, Languages, and Programming (ICALP)*, volume 7391 of *Lecture Notes in Computer Science*. Springer, 2012.

[7] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. *arXiv preprint arXiv:1704.01665*, 2017.

[8] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi. Exact combinatorial optimization with graph convolutional neural networks. *arXiv preprint arXiv:1906.01629*, 2019.

[9] J. Huang, M. Patwary, and G. Diamos. Coloring big graphs with AlphaGo Zero. *arXiv preprint arXiv:1902.10162*, 2019.